

de Oliveira, editors, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, pages 249–280. John Wiley and Sons, Chichester, 1986.

- [18] J.K. Osterhout. *Tcl and the Tk Toolkit*. Addison–Wesley, 1994.
- [19] W. Rachowicz, J.T. Oden, and L. Demkowicz. Toward a universal h - p adaptive finite element strategy. part 3. design of h - p meshes. *Computer Methods in Applied Mechanics and Engineering*, 77:181–212, 1989.
- [20] M.P. Reddy, M.K. Deb, J.M. Bass, and Hui Ning. Numerical simulation of non-conventional wells using adaptive finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 1997. this issue.
- [21] W.W. Tworzydło and J.T. Oden. Towards an automated environment in computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 104:87–143, 1993.

- [6] I. Babuška and M. Suri. The p - and h - p - versions of the finite element method, an overview. *Computer Methods in Applied Mechanics and Engineering*, 80:5–26, 1990.
- [7] R.E. Bank. Analysis of local a posteriori error estimates for elliptic problems. In et al. I. Babuška, editor, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, pages 119–128. John Wiley and Sons, Ltd., Chichester, 1986.
- [8] R.E. Bank and A. Weiser. Some a posteriori error estimates for elliptic partial differential equations. *Mathematics of Computation*, 44(170):283–301, 1985.
- [9] L. Demkowicz, J.T. Oden, W. Rachowicz, and O. Hardy. Toward a universal hp-adaptive finite element strategy. part 1: Constrained approximation and data structure. *Computer Methods in Applied Mechanics and Engineering*, 77:79–112, 1989.
- [10] L. Demkowicz, J.T. Oden, and T. Strouboulis. Adaptive finite element methods for flow problems with moving boundaries. part i. variational principles and a posteriori estimates. *Computer Methods in Applied Mechanics and Engineering*, 46:217–251, 1984.
- [11] W. Gui and I. Babuška. The h , p , and h - p versions of the finite element method in one dimension, part 1. the error analysis of the p -version , part 2. the error analysis of the h - and h - p versions, part 3. the adaptive h - p version. *Numerical Mathematics*, 49:577–612, 613–657, 659–683, 1986.
- [12] A.K. Noor and I. Babuška. Quality assessment and control of finite element solutions. *Finite Element in Analysis and Design*, 1995. to appear.
- [13] J.T. Oden. adaptive finite element methods for problems in solid and fluid mechanics. In R. Voight, editor, *finite element theory and application overview*. Springer-Verlag, New York, 1988.
- [14] J.T. Oden, J.M. Bass, C.Y. Huang, and C.W. Berry. Recent results on smart algorithms and adaptive methods for two- and three-dimensional problems in computational fluid mechanics. *Computers and Structures*, 35(4):381–396, 1990.
- [15] J.T. Oden and L. Demkowicz. Adaptive finite element methods for complex problems in solid and fluid mechanics. In *Computational Mechanics*. ITT, Bombay, India, 1985.
- [16] J.T. Oden and L. Demkowicz. Advances in adaptive improvements: A survey of adaptive finite element methods in computational mechanics. In A.K. Noor and J.T. Oden, editors, *State-of-the-Art Surveys in Computational Mechanics*. A.S.M.E. Publications, N.Y., 1993.
- [17] J.T. Oden, L. Demkowicz, T. Strouboulis, and P. Devloo. Adaptive methods for problems in solid and fluid mechanics. In I. Babuška, O.C. Zienkiewicz, J. Gago, and E.R.A.

Applications of *hp*-adaptive finite element methods go far beyond the few cases presented herein, and include such problem classes as: nonlinear visco-plastic material behavior, nonlinear transient dynamics, computational fluid dynamics (compressible and incompressible), heat transfer, acoustics, electromagnetics and many others. Extensive studies of *hp*-adaptive methods in most of these problem classes support the following observations:

- Residual error estimators provide a reliable measure of the quality of the solution and a robust driving mechanism for automated *hp*-adaptive mesh refinement for linear and nonlinear steady-state problem classes. Moreover, in many such applications the estimated global error provides a reasonable measure of the pointwise errors away from singularities.
- *hp*-Adaptive technology is fully applicable to the solution of nonlinear, time dependent, problems for a broad class of physical models.
- ProPHLEX provides versatile environment for the accelerated development of FEM codes for research and engineering applications.

Acknowledgments

The names PHLEX, ProPHLEX, PHLEXsolid, PHLEXthermal, and UNISIM are trademarks of The Computational Mechanics Company, Inc. X-windows, Motif, Patran, P3/CFD, HyperMesh, and PostScript are trademarks of their respective owners.

References

- [1] M. Ainsworth and J.T. Oden. A unified approach to *a posteriori* error estimation using element residual methods. *Numerical Mathematics*, 65:23–50, 1993.
- [2] I. Babuška and B. Guo. The *hp* version of the finite element method with various elements. *SIAM J. Numerical Analysis*, 25(4):837–861, 1988.
- [3] I. Babuška and W.C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM J. Numerical Analysis*, 15(4):736–753, 1978.
- [4] I. Babuška and W.C. Rheinboldt. A posteriori error estimates for the finite element method. *International Journal for Numerical Methods in Engineering*, 12:1597–1615, 1978.
- [5] I. Babuška and W.C. Rheinboldt. Reliable error estimation and mesh adaptation for the finite element method. In J.T. Oden, editor, *Computational Methods in Nonlinear Mechanics*, pages 67–108. North Holland, N.Y., 1980.

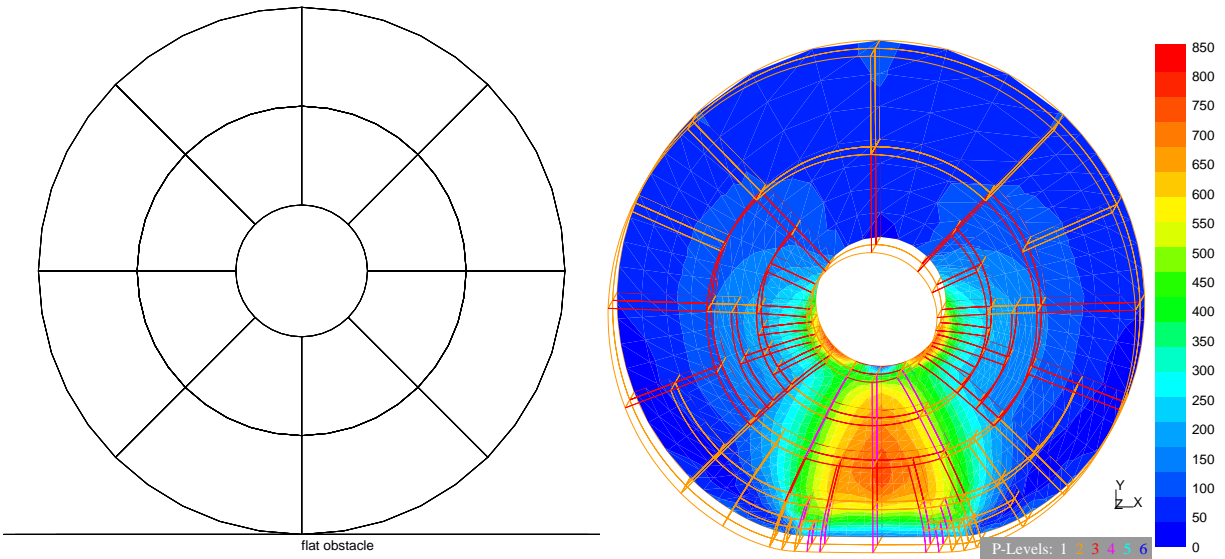


Figure 14: Initial mesh for the cylindrical contact problem.

Figure 15: Cylindrical contact problem: adapted mesh and deformed configuration with von Mises stress.

cheaper solutions, so that real engineering problems can be solved with ProPHLEX on a typical workstation or even on high end PC computers.

In order to further increase the range of application of the ProPHLEX environment, currently efforts are focused extending various aspects of the library including:

- Implementation of additional algorithms in the solver, including an eigen-solver and normal modes computations.
- The capability to mix “standard” (discrete) finite elements with the continuum solver presented in ProPHLEX.
- Providing access to a geometry database, which will allow geometric associativity, resulting in a better description of fine details of the domain which may not be present in the initial, crude mesh.
- Building libraries for specialized applications including; a standardized material database, large deformation subroutines, a rigid surface contact library, and other functionalities.
- Porting ProPHLEX to wider range of computer architectures, including the Windows NT operating system.

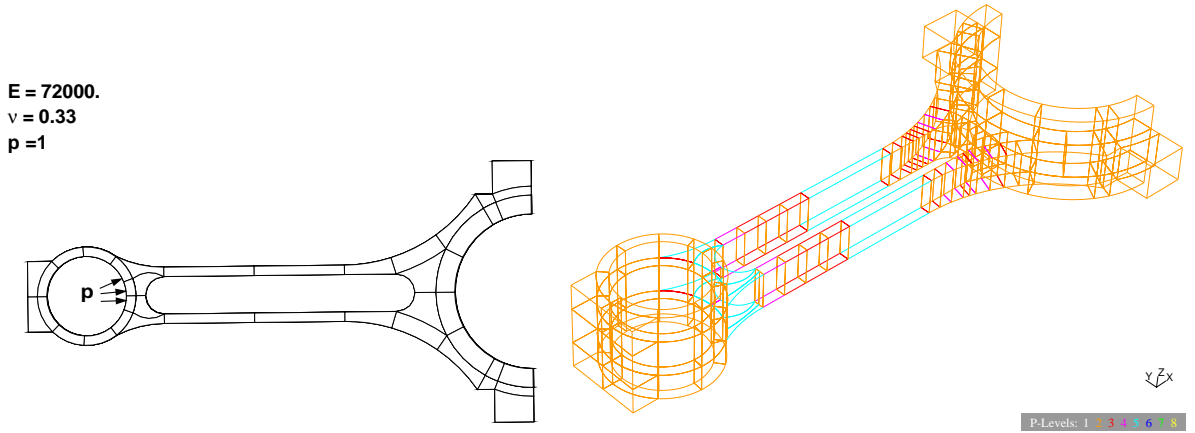


Figure 12: Initial mesh and the loading conditions for an hp -adaptive solution for a connecting rod.

Figure 13: hp -adapted mesh for the connecting rod analysis (p -order up to 5).

Example 3 - Large Deformation Contact Analysis

As final example of an hp -adaptive solution for a nonlinear problem, we present the large deformation analysis of a hollow cylindrical body in contact with a rigid flat surface (the detailed data will not be discussed here). The initial second-order mesh for this problem is shown in figure 14. After moving the contact obstacle towards the cylinder, the problem was solved by combining nonlinear iterations with hp -adaptive mesh refinement passes. The error estimation in this solution was based on a linearized version of the residual method, without any special terms for contact condition. Thus, these error values were only used as indicators driving the adaptive process, not as a measure of the quality of the solution. The resulting final mesh and the deformed configuration, colored by von Mises stress (calculated from the Cauchy stress tensor), are shown in figure 15. Clearly, the hp -adaptive mesh refinement is focusing on resolving the high stress regions in the contact zone.

8 Summary and Future Directions

The basic goal of the ProPHLEX development environment is to allow end-users to build customized applications for various computational mechanics problems, without extensive efforts being spent on basic FEM technology. Shape function computations, numerical integration, linear solvers, and even modern postprocessing don't have to be "reinvented" by every researcher/developer in the field. The addition of hp -adaptivity results in better and

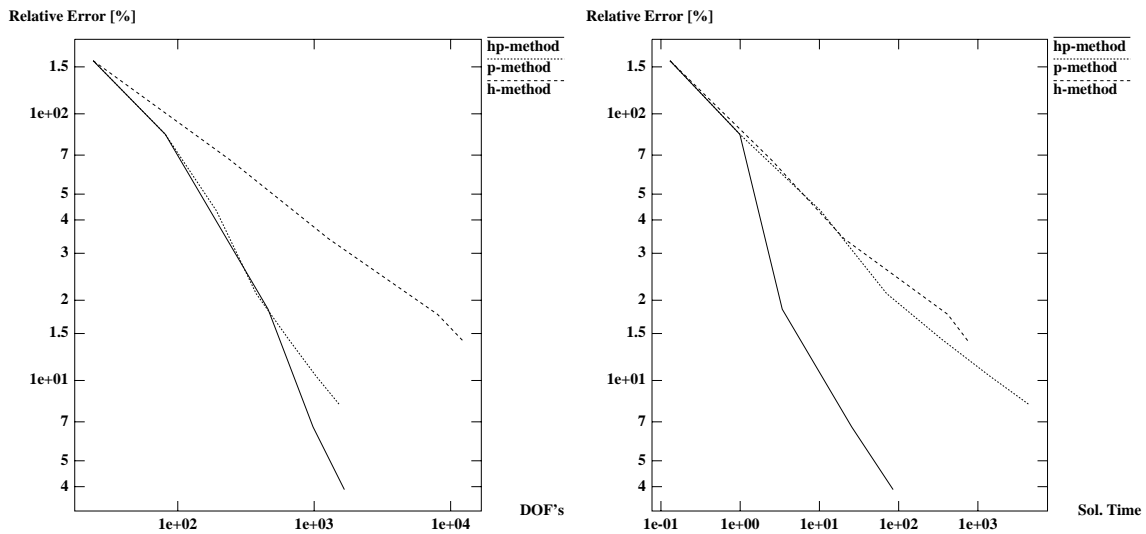


Figure 11: Convergence curves for the cantilever beam problem: (a) error estimate vs. the number of degrees of freedom and (b) error estimate vs. the solution time.

Example 2 - Connecting Rod

This is a problem representative of a typical (although somewhat simplified) geometry of a connecting rod, subjected to a misaligned compressive loading. The overall load pattern and a side view of the initial coarse mesh consisting of 20-noded bricks are shown in figure 12. On this initial mesh, with 3495 degrees of freedom, the finite element analysis predicted a maximum displacement of 0.128 and maximum von Mises stress of 102.4, while the estimated global error was 38 percent.

After 3 adaptive passes, the error was brought below a prescribed limit of 5 percent; the corresponding mesh with a total of 6804 degrees of freedom and p -orders up to 5 is shown in figure 13. The automated mesh adaptation was concentrated mainly in the highly stressed slender connecting elements and in the shape transition zones. On this new adapted mesh the maximum displacement was equal to 0.169, the maximum von Mises stress was 141.6 and the estimated global error was 3.6 percent. The hp -adapted mesh has produced more accurate stresses and displacements while reducing the error by an order of magnitude with only a two-fold increase in the size of the problem.

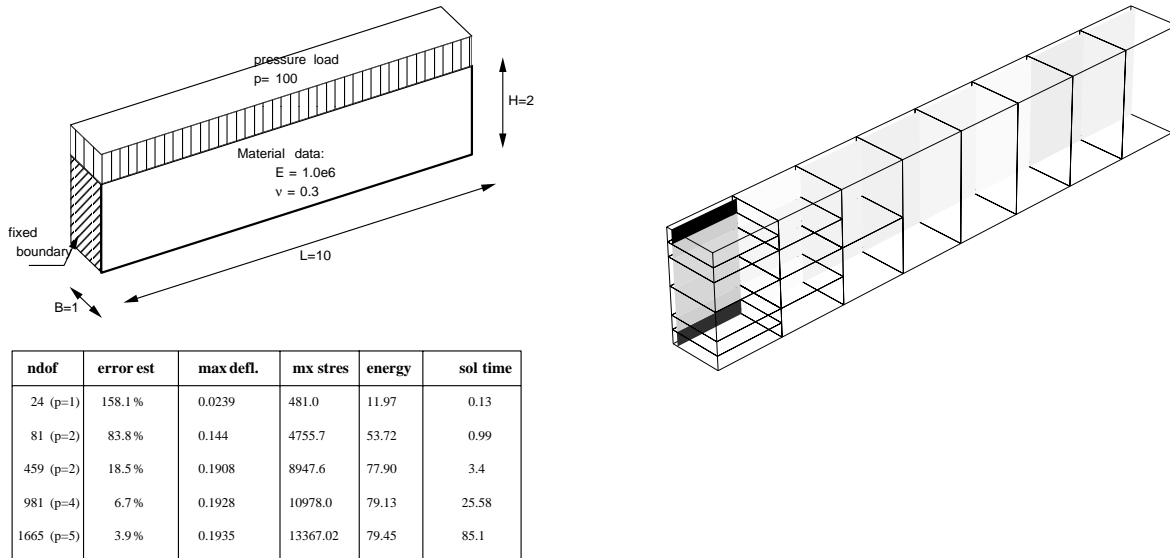


Figure 9: Problem statement and convergence history for the cantilever beam problem.

Figure 10: Final hp -adapted mesh for the cantilever beam problem (p -order up to 5) and a slice colored by element error indicators.

A sequence of mesh refinements produced the hp - mesh shown in figure 10 (due to the black-and-white reproduction, the p -order is not distinguishable in this figure). The maximum order of approximation is $p = 5$ in the longitudinal direction of a beam. The corresponding convergence table is shown in figure 9. One can easily observe that the error in displacements is well within the estimated error bounds. However, the error in the maximum stress appears to exceed the estimated global error level – this is caused by the singular nature of the solution in this case. Importantly, the stress error outside of the singular point is within the estimated bounds.

The above problem was also solved using adaptive strategies limited to h -only or p -only methods (still local and directional). A comparison of their convergence rates with the full hp -method is shown in figure 11. The figure shows two plots: a typical graph of error versus the number of degrees of freedom and a similar graph with the solution time on the abscissa. Clearly, the combined hp -method provides superior convergence rates with respect to both solution time and problem size. Importantly, all of the adaptive methods converge significantly faster than uniform h -refinement or uniform p -enrichment.

Variational Formulation (virtual work): Assume a virtual displacement or variation $\mathbf{v} = \delta \mathbf{u}$, the virtual work will be:

$$\int_{V_0} \frac{\partial}{\partial a_j} \left(S_{j\ell} \frac{\partial x_i}{\partial a_\ell} \right) \delta u_i dV_0 + \int_{V_0} \rho_0 F_{0i} \delta u_i dV_0 = 0$$

Integrating by parts, using the Gauss divergence theorem and renaming our variation $\delta \mathbf{u} = \mathbf{v}$ we get in a closed form

$$f(\nabla \mathbf{u}) = \int_{V_0} (S \nabla_{\mathbf{x}} \nabla \mathbf{v} - \rho_0 F_0 \cdot \mathbf{v}) dV_0 - \int_{S_{0\sigma}} \mathbf{T}_0^n \cdot \mathbf{v} dS_0 = 0 \quad (7.9)$$

As the problem is nonlinear, therefore we use the Newton algorithm to solve it:

$$\left[\frac{\partial f(\nabla \mathbf{u})}{\partial \nabla \mathbf{u}} \right] \Delta(\nabla \mathbf{u}) = -f(\nabla \mathbf{u}) \quad (7.10)$$

so, finally, equations 7.9–7.10 in ProPHLEX notation become:

$$\int_{V_0} A_{ij}(k, n) \frac{\partial v_k}{\partial a_i} \Delta \left(\frac{\partial u_n}{\partial a_j} \right) dV_0 = \int_{V_0} F_i(k) \frac{\partial v_k}{\partial a_i} dV_0 + \int_{V_0} F(k) v_k dV_0 + \int_{S_{0\sigma}} G(k) v_k dS_0 \quad (7.11)$$

where,

$$\left\| \begin{aligned} A_{ij}(k, n) &= S_{ij} \delta_{kn} + D_{i\ell\alpha j} \frac{\partial x_n}{\partial a_\alpha} \frac{\partial x_k}{\partial a_\ell} \\ F_i(k) &= -S_{i\ell} \frac{\partial x_k}{\partial a_\ell} \\ F(k) &= \rho_0 F_{0k} \\ G(k) &= T_{0k}^\nu \end{aligned} \right.$$

Example 1 - Cantilever Beam

The first example is that of an *hp*-adaptive solution of a slender body representative of a cantilever beam loaded with a distributed pressure on the top surface. The problem statement for this study is shown in figure 9. The solution process was started from a *single linear finite element* with a requested target global error level below 5 percent.

here on detailed studies of effectivity indices, convergence rates etc. but rather examples are presented which illustrate the overall performance of the method.

7.1 Solid Mechanics Applications

Large Deformation Elasticity Problem

We present here a simplified version of a large deformation elasticity problem formulated in a weak form suitable for solution using ProPHLEX. Note that the PHLEXsolid application also includes a fully nonlinear material formulation, with time effects due to viscosity and inertia, as well as an accurate friction model for the contact surfaces.

The Differential Equations: The partial differential equations which describe the equilibrium of a three-dimensional body undergoing large deformations is given by

$$\frac{\partial}{\partial a_j} \left(S_{j\ell} \frac{\partial x_i}{\partial a_\ell} \right) + \rho_o F_{oi} = 0. \quad (7.7)$$

Here, $S_{j\ell}$ represents the components of the 2nd Piola–Kirchhoff stress (which is symmetric), a_i are referential coordinate components, x_i are the components of the current coordinates of a material particle ($\mathbf{x}_i = \mathbf{a}_i + \mathbf{u}_i$ where \mathbf{u}_i are the displacements of a particle), ρ_o is the mass density per unit volume in the undeformed configuration, and F_{oi} are the components of the body forces per unit volume in the undeformed state. The relationship between the surface traction $\overset{n}{T}_{oi}$ and the stress tensor given by

$$S_{j\ell} n_{oj} \frac{\partial x_i}{\partial a_\ell} = \overset{n}{T}_{oi}. \quad (7.8)$$

Note $\overset{n}{T}_{oi}$ are components of the surface traction in the reference configuration which have the same direction as those in the current configuration.

Boundary Conditions: Typical boundary conditions are expressed with tractions in the referential coordinates, as indicated in Eq. (7.8). Thus,

$$\begin{aligned} u_i &= \hat{u}(x, y, z) && \text{on } \Gamma_u \\ u_n &= u \cdot n = \hat{u}_n(x, y, z) && \text{on } \Gamma_c \\ t_{\tau_1} &= t_{\tau_2} = 0 \\ t_i &= \overset{n}{T}_{oi}(x, y, z) && \text{on } \Gamma_t \end{aligned}$$

where Γ_u, Γ_t , are boundaries with prescribed displacements and tractions, respectively, while Γ_c is the frictionless contact boundary.

The *element hopping* algorithm is complicated significantly by adaptivity: due to refinement, an element can have one or two neighbors across any face, or it can be connected to half of the face of a “big” neighbor. As the initial mesh of elements is fully unstructured, the number of possible configurations may be quite large, based on an internal numbering of the faces and on the rotation of one element with respect to another.

The particle trace generator in ProPHLEX uses ‘exact’ integration within each element (using an adaptive Runge–Kutta procedure) and then uses the above *element hopping* algorithm to cross the interelement boundary and continue. Accurate numerical integration is very important for high order elements, where the particle trace may actually loop within single element.

All of the figures in the following section are produced using the built-in ProPHLEX postprocessor.

7 Applications and Numerical Results

ProPHLEX has been used at COMCO as the basis for building a number of application codes for various areas of applied mechanics problems:

- **PHLEXsolid**, can solve a multitude of solid mechanics analysis problems, including large deformation mechanics of three-dimensional bodies with linear or nonlinear material properties (including visco–plastic effects). It has static and dynamic options which can be coupled with rigid surface contact for arbitrary surface shapes.
- A specialized version of the solids code includes algorithms to accurately solve pseudo-dynamics problems related to steady-state rolling contact with friction.
- **UNISIM**, developed jointly with TEXACO, models one and two–phase porous media immiscible flows, including the unique capability to accurately analyze the flow in the vicinity of nonstandard wells (horizontal, lateral, etc.) [20].
- **PHLEXthermal** is an application used to analyze linear and non-linear heat conduction problems. The source code for the PHLEXthermal routines are part of ProPHLEX development environment.
- For various in–house projects we have also developed general purpose CFD codes for the solution of compressible and incompressible flow problems. Based on the previous version of the ProPHLEX kernel, **P3/CFD** was marketed by PDA Engineering in the suite of Patran programs. Specialized versions are also used in the analysis of non-Newtonian flows for various flow regimes.

In this section, selected numerical examples of residual error estimation and *hp*-adaptive mesh refinement strategies are presented from the PHLEXsolid application. Our main objective is to illustrate the practical performance of these methods as an automated aid to producing high-quality finite element results, starting from a very coarse initial mesh and without the necessity of hand-designing fine problem-specific meshes. Thus, we do not focus

- Start with an arbitrary element (usually the one used in the last search is a very good guess).
- Using a triangular approximation of all faces of the element check if the line segment from the centroid of the element to the point P is crossing any of the faces.
- If the line crosses any face then find the neighbor across this face and repeat the previous test.
- Once the element presumably containing the point P is found, we have to check it using the true geometry of the element. This requires iterative inversion of the isoparametric map and may fail if the point is far outside the element (This may happen for long, thin, curved elements, for which the 'centroid' is far from the actual center of the map).
- If the point P is outside the element, find the correct neighbor and repeat the search.
- If any of the searches cross an actual boundary, then the whole process has to be repeated with initial node found using the nearest boundary element.
- If the search fails again, then the point is presumed to be outside of the domain boundary.

```

      subroutine POSTF ( el2get, iloc, isol, u, uxyz, unew )
c*****
c*      Parameters:
c*      el2get  In - element number to get data for
c*      iloc    In - entry position in unew to fill
c*      isol    In - component name to be loaded (macro)
c*      u       In - solution vector at a point for el2get
c*      uxyz    In - gradient of u (if requested)
c*      unew    Out - vector of post processing quantities
c*****
      INTTYPE iloc, isol, el2get
      REALTYPE u(*), uxyz(MAXNCOMP,3), unew(*)
c+ INTTYPE and REALTYPE are host-dependent, usually INTEGER*4, REAL*8
c+ when passed to C code: 'INTTYPE' is 'int', 'REALTYPE' is 'FORTREAL'
      .....
-----
      call PH_REGISTER_FUNCTION_(FUNC_POST,postf)
-----
#define F_ FORTREAL
/* macro FORTREAL is host-dependent: double or float */
void function POSTC ( int* el2get, int* iloc, int* isol,
      F_ * u, F_ uxyz[3][MAXNCOMP], F_ * unew );
-----
      PH_RegisterFunction(FUNC_POST, postc);

```

Figure 8: FORTRAN function header defining derived components, its registration, and respective C counterparts

Visualization and postprocessing

At present, none of the commercially available postprocessors are capable of visualizing solutions obtained from an hp -adaptive FEM program. ProPHLEX can produce acceptable results files in Patran and HyperMesh formats, but the accuracy of high order p -adapted elements is reduced due to the necessary process of substituting them by groups of simple cubes (*dices*). The ProPHLEX postprocessor does essentially the same job by dicing elements into smaller ones, but it does not lose any information in this process, as the quality of the approximation can be adjusted according to the user's preferences and the picture redrawn with an appropriate amount of detail. All other information (presence of the boundary, boundary conditions, current order of approximation, neighbor information) is preserved and can be drawn or taken into account during visualization.

ProPHLEX comes with a quite general set of interactive postprocessing capabilities. It works on any workstation with X-Windows, but if available it uses hardware graphics (OpenGL or Starbase standards), and it comes also with a software implementation of OpenGL (public domain Mesa library, for high quality plots on a standard X terminal). All plots can be stored to a file for hardcopy purposes as a PostScript file or a screen dump (RGB files in OpenGL/Mesa).

In addition to displaying the hp -adapted mesh, the ProPHLEX postprocessor provides the following standard visualization features:

- arbitrary slicing planes,
- isosurfaces of any component colored by any other component,
- xy plot of the solution between two points,
- interactive solution probe,
- plots of boundary conditions and loads,
- deformed configuration plots,
- velocity vectors,
- particle traces.

for any arbitrary component of the solution. A developer registered routine (see Figure 8) is used to compute secondary components (eg. stresses in solid mechanics or Mach number in CFD application). All capabilities of the postprocessor can be selectively activated, depending on the problem (solid mechanics uses a deformed configuration plot while fluids applications use particle traces and velocity vectors). Optionally, the developer can add problem specific plots, eg. for the PHLEXsolid it is possible to display contact surfaces and membranes. This is again, where registration of various user defined structures and functions is applied.

Particle traces and the solution probe use internal database information about connectivities between elements to provide fast searches for an arbitrary location in space. The search procedure for an element containing given point P is based on an *element hopping* algorithm:

ProPHLEX includes a state-of-the-art, multifrontal, sparse solver which is highly optimized. It can be used either in a fully in-core or in an out-of-core mode. Sparse solvers are integrated with the rest of ProPHLEX through a flexible interface which readily allows for the linking in of any alternative sparse solver implementation best suited for the underlying hardware. This interface allows its clients to use customized sparse solvers in complex setups in an optimal manner.

6 Graphics and Post Processing

User Interface

ProPHLEX is designed to be run in batch as well as in interactive mode. In batch mode the code is controlled by a script written in `tc1` [18] language. All major steps of the solution process, including control of all `PARAMETERs` and saving the results in various formats, can be performed in this mode without initialization of the X/Motif interface. The batch processing can also be performed after this initialization, with full control of most postprocessing options (manipulation of views, activating various components, creating isosurface or slice plots, etc). This allows postprocessing of the results and the creation of multiple hardcopy/screen dumps from the solution process, which is especially useful for nonlinear iterations and time-dependent problems. The output files may be used in creating demonstrations as movie files or postprocessed using various commercial postprocessors.

The majority of users, especially during the development of an application, will use ProPHLEX in the interactive mode, which includes access to all steps of the solution (linear/nonlinear solvers, mesh modifications, error estimation and automatic adaptation, creation of results files, etc.) and general postprocessing capabilities. Except for the main menu bar, and main postprocessor window, all 'forms' are optional and are easily customizable. A user can select which ones of the preprogrammed forms are to be used, and arbitrary user-defined forms can be added. All standard Motif widgets are accessible, but the user can (and is encouraged to) use a simple GUI builder, facilitating the use of most standard elements of the interface (buttons, input/output fields, sliders, etc.).

To assure proper usage of X/Motif, and to separate the ProPHLEX library from the developer's code, the registration mechanism described in section 4 is used. Similar registration is also used to register various data structures used for the GUI and postprocessing. Structures are used especially for the customization of the postprocessor to:

- select the solution components for the postprocessor,
- select the functions of the postprocessor which should be active,
- select which components are available at the current stage of the solution process.

evaluate boundary functional, material routines), and global solution sequence, are written in FORTRAN, while functions required for postprocessing and visualization are written in C. It is possible to write all of these functions in C (and not use FORTRAN entirely),

Most of the registered functions may be dynamically 'reregistered' during the solution process. A natural example of such a reregistration is used in the solution of incompressible fluid problems. Each stage of the computation (velocity step, energy and pressure step) is solved independently, and thus the calculation of the weak form coefficients (and appropriate boundary conditions) is performed in different routines. Various MCOEFF/BCOEFF routines are reregistered at the beginning of each stage. This provides a cleaner code and is more efficient than writing single routine with a giant switch statement based on the current stage of the computation.

5 Linear Equation Solver

ProPHLEX is equipped with various optimized linear equations solvers that can be interfaced flexibly with the rest of the kernel. An application can register the particular linear solvers it may need through the regular API (applications programming interface). Through a common interface shared by all solvers, an application can control the actions of a solver without requiring detailed knowledge about the specific solver that may be selected by a user at runtime (through the user interface). Also, these solvers are capable of handling *hp*-constraints (i.e. the ones generated whenever a mesh is *hp*-adapted).

The present suite of linear solvers ProPHLEX consists of:

- iterative solvers:
 - Generalized Minimum Residual (GMRES),
 - Conjugate Gradients (CG),
- direct solvers:
 - Multifrontal Sparse,
 - Frontal,
 - Skyline.

A suite of pre-conditioners including diagonal and block diagonal preconditioners is available with the iterative solvers. Any of the direct solvers can be also be used to compute specialized preconditioners. The application users may specify the matrix-vector product operation in iterative solvers to be performed in a number of different ways depending upon the optimization required. The available options are:

- in a matrix-free manner where the element level stiffness matrices are never formed explicitly,
- block assembled matrix-vector product,
- fully assembled sparse matrix-vector product.

```

OBJECT BC_Kinematic {
  int      objname ;
  OBJECT   Any_Object * CoordFramePtr;
  FORTREAL Values[MAX_SOLID_BCVALS];
  OBJECT   GenData * LdFun[3];
  UBYTE    DirFlags[3];
  UBYTE    GorEflag;
  UBYTE    CoordType;
  char *   CoordFrameName;
};
-----
  static char * displacementlabels[] = { "Components", "", "",
                                         "Flags", "", "" };
  PH_GClass_Register( GCLASS_BC, "Displacement", displacementBC_create,
                    6, displacementlabels );
-----
static void displacementBC_create( const char * class_instance_name,
  PTR * class_instance,
  OBJECT GenData ** data )
{
  /* consistency tests deleted */
  OBJECT BC_Kinematic * bcobj;
  PH_Obj_create( (PTR)&bcobj, OBC_Kinematic );

  for ( i=0; i<3; i++ ) {
    bcobj->Values[i]   = PH_GData_GetValue( data[i], NULL);
    bcobj->DirFlags[i] = PH_GData_GetValue( data[i+3], NULL );
  }
  bcobj->CoordType     = COOR_GLOBAL;          /* global by default */
  bcobj->GorEflag      = 1;                   /* centroid value */

  for ( i=0; i<6; i++ )
    if( data[i] ) PH_GData_Delete( data[i] );
  * class_instance = (PTR)bcobj ;
}
-----
  EXTERNAL bcldkin, drldkin
  call PH_REGISTER_BC_ ( 'Displacement', bcldkin, drldkin )
-----
Displacement left_wall { Flags = 0 1 1, Displacement = 0 0 0.1 }

```

Figure 7: Simple boundary OBJECT, its registration code, creator function, registration of associated functions (FORTRAN code), and input example.

A typical application developer usually defines several specialized classes. For example, the following MATERIALs are defined in PHLEXsolid: `IsoHook`, for linear elastic materials, `Mooney` for rubber-like materials, etc. Each of these classes will have a different data OBJECT contents, different fields for the input and different functions (like *methods* in C++) performing necessary computations, typical for each class. All OBJECTs in a Material class additionally have the ability to register special functions, which in PHLEXsolid is used to compute stresses from strains. Each OBJECT in the Boundary and Point_Load classes has two functions: to compute the contribution to the boundary integral and to define the kinematic boundary conditions.

Figure 6 shows a simplified definition of the Brick OBJECT (which is a generic hexahedral element used in ProPHLEX). Figure 7 shows the definition of an OBJECT used to store a simple boundary condition, and the code required to register all functionality related to this OBJECT. `LdFun` is used for a load multiplication factor dependent on the time (for a non-linear or dynamics simulation). The actual definition of the `LdFun` function is defined by the user in the input data and interpreted when needed. This generic function capability is an additional feature in ProPHLEX and can be used to define an arbitrary stress-strain relation, complex inflow profile, temperature distribution on the boundary, dependency of viscosity on temperature, etc. In all of these cases the actual definition of the function is supplied through the input data, where the actual user of the code may define the functions as one or two-dimensional tables, or as arbitrary expressions (in the `tcl` language).

Function Registration

In an Object Oriented programming environment, the preferred way to introduce developer required functionality, is by using *methods* in a derived classes. A global functionality (eg. a solution sequence) can be introduced as a method in a global object (usually a single instance of the object is created; examples of such objects may be called Mesh, Solution, etc.). Local functionality can be introduced as methods in other classes of objects (eg. material properties or element stiffness calculations can be methods for each class of Element objects).

In ProPHLEX the same technique is used to provide a clear subdivision between the general part of the code (library) and developer's defined specialization. Obviously the languages used in the code do not provide this functionality, and it has been 'manually' provided as a set of functions used to 'register' addresses of various routines (C functions or FORTRAN subroutines). This registration is used on a global level: to register a solution process, or a preconditioner for an iterative solver, as well as on the OBJECT class level: to register a transformation function for the Coordinate_System class, or boundary evaluators for the Boundary class (see Fig. 7).

Most of the registration can be performed alternatively from FORTRAN or C code, and the registered functions may be written in either language. Traditionally the element routines (MCOEFF - to evaluate the volume integral coefficients of the weak form, BCOEFF - to

Developer Defined OBJECTs

While the ProPHLEX data base is written mainly in C (with minimal FORTRAN), many functionalities present in C++ or other Object Oriented (OO) languages are clearly visible within the code. All OBJECTs can be treated as derived from a general class of objects equipped with save/restart, browsing, and various containers. Note that automatic save/restart capabilities, especially for pointer fields, is not provided by current OO languages.

```
#define OBJECT struct
#define UBYTE unsigned char
#define NDIM 3
#define NFACE 6
#define NGNODE 8

OBJECT Brick {
    int          objname;          /* internal info */
    int          ElemID;          /* from input file if defined */

    OBJECT Brick *  father;       /* genealogical information */
    UBYTE         Breaks [ NDIM ];
    UBYTE         nsons;
    OBJECT Brick ** sons;         /* dynamically allocated array */
                                /* of pointers (of size nsons) */

    OBJECT Brick *  neigh [ NFACE ]; /* neighbors across faces */
    OBJECT Gnode *  gnodes [ NGNODE ]; /* vertices of hex */
    OBJECT Pstruct * p_struct;     /* if element enriched */
    OBJECT BCelem * BCinfo;       /* boundary info */
    OBJECT ErrEst * ElemErr;      /* estimated error */
    OBJECT CIO *   material;      /* material or property */
    OBJECT Any_Object * appData;  /* for expansion by developer */
    .....
};
```

Figure 6: Simplified definition of Brick OBJECT, *#define*'s are provided for clarity, they appear in different source files.

There are also a number of other aspects related to OO technology that are clearly visible in the ProPHLEX user defined OBJECTs. For example, they are all registered as classes within predefined global classes of OBJECTs. Among the predefined classes are: Material, Property, Boundary, Point_Load, Coordinate_System, and generic User_Data OBJECTs. Each of these is 'derived' from a general class, which (in addition to all the functionalities presented above) has built-in functions to read in free format data from the input file, and a textual name (used and interpreted by the kernel eg. to assign material data to Elements).

with a need to verify the local mesh consistency at each stage of adaptation (for a review of various database designs used with adaptive FEM codes see [9]), requires that the database use dynamically allocated `struct`'s (from C language). These structs are called OBJECTs, and they are designed to contain user data as well as all connectivities between elements, nodes, degrees-of-freedom etc. For efficiency, the connectivities are stored as memory addresses (pointers) and interface functions are provided which allow FORTRAN access.

The textual definition of OBJECTs (similar to standard C code) is preprocessed by a proprietary program `dsmake`, based on the standard UNIX utilities `lex` and `yacc`. This program produces a small set of functions for each OBJECT providing save/restart capabilities, containers for the OBJECTs (lists, queues, arrays, sets) with predefined iterators, a browsing capability (without speed penalties or the use of a specialized debugger), and access to all OBJECT fields from FORTRAN routines. The set containers are equipped with a full list of Boolean operators. The element sets can be defined interactively (eg. through a bounding box from the postprocessor window) and are automatically adjusted to changes in the mesh due to the adaptive process.

Save Restart Capability

As the database can significantly change during the solution process, it is a non-trivial task to provide the ability to save the state of the solution. In a traditional FEM code, all that is required to continue the solution is to retrieve the values of the solution array, while in an adaptive database, like in ProPHLEX, there is much more information to be stored. ProPHLEX OBJECTs are built with an automatic save/restart capability: each OBJECT can be written to an external file (even the fields containing pointers to other OBJECTs), and retrieved later. As the file is written optionally as plain text (in ASCII mode), it can be read on different computer platforms, which allows for example the solution of the problem on a supercomputer, saving the results to a file and then retrieving the complete state of the process on a workstation for convenient visualization.

The save/restart capability has a powerful ability to adjust to certain changes in the code: OBJECTs can be deleted, new OBJECTs defined, existing OBJECTs may have some fields removed or added, even some fields can change the type (eg. between float, double, int, and byte), and ProPHLEX can still read the save file created by previous versions of the code, retrieve all existing information and provide simple default values for the new data. This backward compatibility is very useful and important in a commercial as well as a research development environment. It is provided automatically for all predefined OBJECTs and for all new OBJECTs defined by a ProPHLEX application developer. The developer can define up to 256 arbitrary OBJECTs, mainly for the material, boundary and similar properties, but arbitrary data can also be stored. For example, in the `PHLEXsolid` code there are special OBJECTs to hold information about contact surfaces.

At this point it is customary to exploit linear problems by separating any integrands in $\mathcal{L}(\cdot, \cdot)$ not containing e and move them to the right side of the equation.

$$\mathcal{L}(e, \mathbf{v}) = \mathbf{F}(\mathbf{v}) - \mathcal{L}(\hat{\mathbf{u}}, \mathbf{v}) \quad \forall \mathbf{v} \in H(\Omega) \quad (3.6)$$

This equation is recognizable as a weak formulation for the analytic error in the problem and its right side is the weak form of the residual.

Error estimation, therefore, seeks to find an approximate solution to this problem. In order to reduce the computational cost of this process, a series of element-wise problems are solved instead of a global problem. This is accomplished by writing the error formulation for each element and then using a Lagrange multiplier to impose an interelement continuity constraint on the error. These constraints take the form of interelement boundary flux integrals. As the (analytic) flux is unknown, it must be approximated in some manner – usually using the known approximate solution $\hat{\mathbf{u}}$. A significant amount of research has been done on techniques for calculating these fluxes. The work of Ainsworth and Oden [1], for example, has shown that when calculated the proper way, the resulting estimates provide an upper bound on the analytic error. In the interest of computational efficiency (and only a modest loss of accuracy) our estimates compute the fluxes by using the average of an element’s approximate flux and its neighbor’s.

In creating an approximate solution to the error problem, an important step is the selection of the subspace of $H(\Omega)$ in which to seek the error estimate. If the approximate solution $\hat{\mathbf{u}}$ is a finite element solution the estimate subspace must be selected so that it is significantly different from the solution subspace. In practice, this estimate subspace is constructed using a set of basis functions consisting of polynomials that are higher order than those of the solution subspace and are constructed so that they are nearly orthogonal to the solution subspace (in the sense of an inner product associated with $\mathcal{L}(\cdot, \cdot)$). Obviously, the number and complexity of these basis functions greatly affect the computational cost of the error estimates. In practice, we have found that using a set of functions that are one polynomial order higher than the solution usually provides the best balance between accuracy and cost.

Finally, the above discussion has been presented in terms of a linear problem formulation. These techniques can provide meaningful error estimates for non-linear problems if care is taken in formulating the error problem. In general, a formulation much like Equation 3.6 results with the standard weak form of the residual on the right hand side but a different formulation is used on the left hand side.

4 Object Oriented Data Base

OBJECTS

The ProPHLEX database is significantly more complex than that required for traditional finite element codes. The ability to handle dynamic changes in the mesh (due to adaptivity),

purposes let's examine a simple situation with three elements containing an irregular node along a common edge, (see Figure 5). Assume that the mesh is initially of any uniform order p , and one of the elements is enriched. As the order of approximation on edge (3–4) changes, edge (3–5) must be modified to maintain a continuous solution. Then edge (3–5), and (4–5) must be enriched to the same polynomial order. The database must detect this propagation of enrichment along the constrained edge, and enrich all affected elements. Otherwise, the continuity of the solution will not be preserved.

hp Constraint on an Edge

The actual enforcement of an hp constraint is performed in the same manner as with h -adaptivity:

$$U_u = R * U_c$$

where U_u is the unconstrained vector, U_c is the constrained vector, and R is the constraint matrix. The only difference here is that the constraint matrix R is significantly more complex due to the required continuity of the higher order approximation. If the edge has an order of $p = 3$, for example, then the constraint matrix has 5 rows and 4 columns (4 active dofs: two corners and a double p -dof; 5 constrained dofs: two double p -dofs and a linear midpoint dof). The assembly process is analogous to h -adaptivity.

Error Estimation

ProPHLEX uses the element residual technique to estimate errors in finite element solutions. This technique is based on the early work of [8] and [17] and measures the amount that a particular solution fails to satisfy the governing partial differential equations and uses this function in the solution of a series of small problems to obtain an approximation of the error in the solution. In particular, ProPHLEX uses a variation of this technique which utilizes the weak form of the governing equations.

In formulating a finite element problem, the governing equations are usually expressed in their weak form by multiplying the partial differential equations by a test function \mathbf{v} , integrating over the problem domain and then integrating some of the terms by parts using Green's identity. Finally, at least some of the boundary integrals are replaced by the appropriate weak forms of the boundary conditions. Using this procedure, the finite element problem can be written abstractly in its weak form as: Find $\mathbf{u} \in H(\Omega)$ such that

$$\mathcal{L}(\mathbf{u}, \mathbf{v}) = \mathbf{F}(\mathbf{v}) \quad \forall \mathbf{v} \in H(\Omega) \tag{3.4}$$

The error in an approximate solution $\hat{\mathbf{u}}$ is defined as $\mathbf{e} = \mathbf{u} - \hat{\mathbf{u}}$. We obtain a formulation for the error by substituting $\mathbf{e} + \hat{\mathbf{u}}$ for \mathbf{u}

$$\mathcal{L}(\mathbf{e} + \hat{\mathbf{u}}, \mathbf{v}) = \mathbf{F}(\mathbf{v}) \tag{3.5}$$

of these p -nodes can be independently enriched to any order up to 8 (actually, the highest order used in the solution process is 7, which allows for additional enrichment during the computation of error estimates). In practical cases, the order of approximation is limited to 4–5.

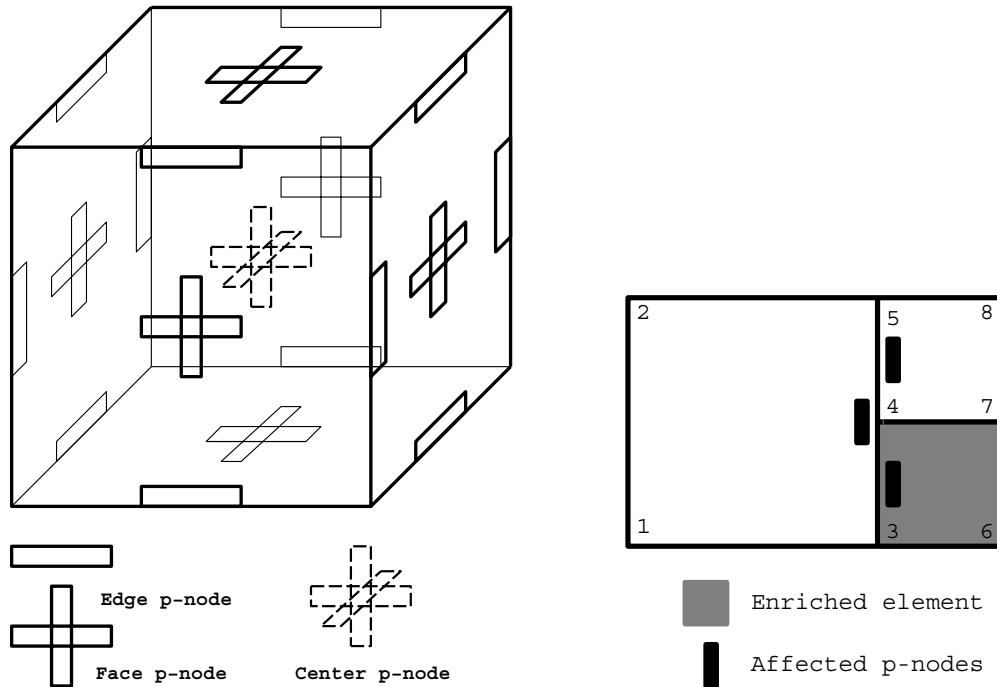


Figure 4: p -Nodes in a Three-Dimensional Element

Figure 5: p -Enrichment of an h -Irregular Mesh

h - p -Adaptivity

h - p -adaptivity combines both h - and p -adaptivity, so that elements can be refined (unrefined), and enriched (unenriched), according to a given adaptive strategy. Most of the algorithmic problems (mentioned above) for h -adaptivity, and p -adaptivity are also encountered collectively with h - p -adaptivity but in a somewhat more complex setting. Some additional issues worthy of mention are outlined below.

Continuity Requirements

As in h -adaptation, only 1-irregular meshes are allowed (two-to-one rule); therefore, additional constraints have to be enforced to ensure continuity of the solution. For illustrative

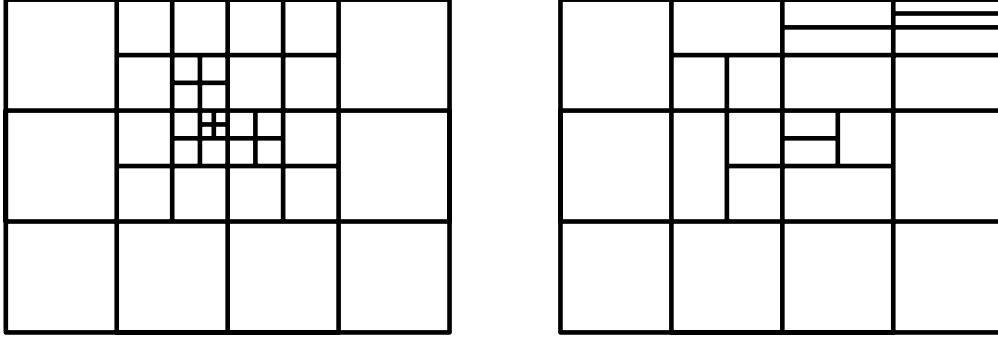


Figure 3: Meshes with Isotropic and Anisotropic Refinements

begins with the standard linear Lagrange shape functions used with h -adaptive techniques. Enrichment of the solution space is obtained by adding new shape functions to the existing ones in such a way that enriching from order p to $(p + 1)$ does not change the existing shape functions or degrees of freedom, but simply adds new ones. This contrasts with traditional Lagrange elements, where adding a $(p + 1)$ node requires moving all the other nodes (except corners) and redefining all of the shape functions.

Using hierarchical shape functions, an approximation of the unknown solution is obtained in the usual form:

$$U_h = \sum_i U_i * N_i = U * N$$

except that the degrees of freedom corresponding to higher order shape functions (i.e., to quadratic, cubic, etc.) cannot be interpreted as values of the solution at a distinct node. Instead, they are values of higher order derivatives of the solution at the midpoints (or linear combination of these derivatives).

In one-dimension, the hierarchical shape functions are defined as follows (associating all new degrees of freedom with the center point of the element).

$$\begin{aligned} N_1 &= \frac{1}{2}(1 - \xi) \\ N_2 &= \frac{1}{2}(1 + \xi) \\ N_c^p &= \frac{1}{p!}(\xi^p - 1), \text{ for } p = 2, 4, 6, \dots \\ N_c^p &= \frac{1}{p!}(\xi^p - \xi), \text{ for } p = 3, 5, 7, \dots \end{aligned}$$

The one-dimensional concepts can be easily extended to two and three dimensions by defining appropriate tensor product spaces. The p -degrees of freedom are now associated with directional derivatives at midpoints of the element edges and with mixed derivatives at the center point of the element. Figure 4. shows the position of p -nodes (i.e., nodes associated with p -degrees of freedom) for a three-dimensional element. In ProPHLEX each

h or p) can offer a wide improvement in the computational effort over more conventional isotropic schemes. This is primarily true because anisotropic adaptation allows for selected refinement and/or enrichment only in the directions of interest (i.e. directions of high error). Thus, anisotropic adaptation may greatly reduce the total number of unknowns in a given problem, thereby reducing the overall required computational effort.

***h*-Adaptivity**

We begin this discussion with an overview of possibly the most widely used class of adaptivity, h -adaptivity. h -adaptivity is a methodology whereby the computational mesh density is increased by replacing a single element with two or more smaller ones. (h stands for the element or mesh size, thus h -adaptivity means reducing the size of the elements). Initial versions of this approach [19, 13] divided each element in four smaller elements (eight in 3D), which tend to produce more uniformly graded, isotropic meshes (see Figure 3a), but usually with many more elements than the anisotropic philosophy adopted here. Anisotropic adaptivity always splits an element into two 'sons' in a selected direction (see Figure 3b).

For cube/quad elements, independent of which strategy is selected, h -adaptivity immediately violates the traditional 'one-to-one' element connectivity rule, with the two new 'sons' having one big neighbor along the common boundary (edge or face). This irregularity can be avoided by utilizing triangular (transition) elements and subdividing the big neighbor, or by defining 'Green' transition elements. The geometric element irregularity can also be accepted and the solution continuity subsequently enforced by adding special algebraic constraints. This latter approach, although somewhat more computationally expensive, has been adopted as the basis for h -refinement in ProPHLEX.

In closing out this overview of h -adaptivity, we note that ProPHLEX enforces the 'two-to-one' rule limiting the number of neighbors across a single edge or element face to two. This limitation minimizes constrained node propagation, which is a special problem encountered with anisotropic adaptivity, as well as the non-physical, geometric stiffening of the solution typical of adaptive rules which allow large numbers of 'slave' elements attached to a single 'master' element.

***p*-Adaptivity**

Contrary to h -adaptivity, p -adaptivity assumes that elements of the mesh remain fixed (in both size and position), but the order of polynomial approximation within each element may change. For p -adaptive schemes the space of the shape functions is 'enriched' by adding higher order functionals. This technique is sometimes referred to as a 'spectral' method, as adaptivity is obtained by increasing the spectral order of the approximation of the solution.

The implementation of p -adaptivity in ProPHLEX is based upon the concepts embodied in hierarchical shape functions. The class of hierarchical shape functions used in ProPHLEX

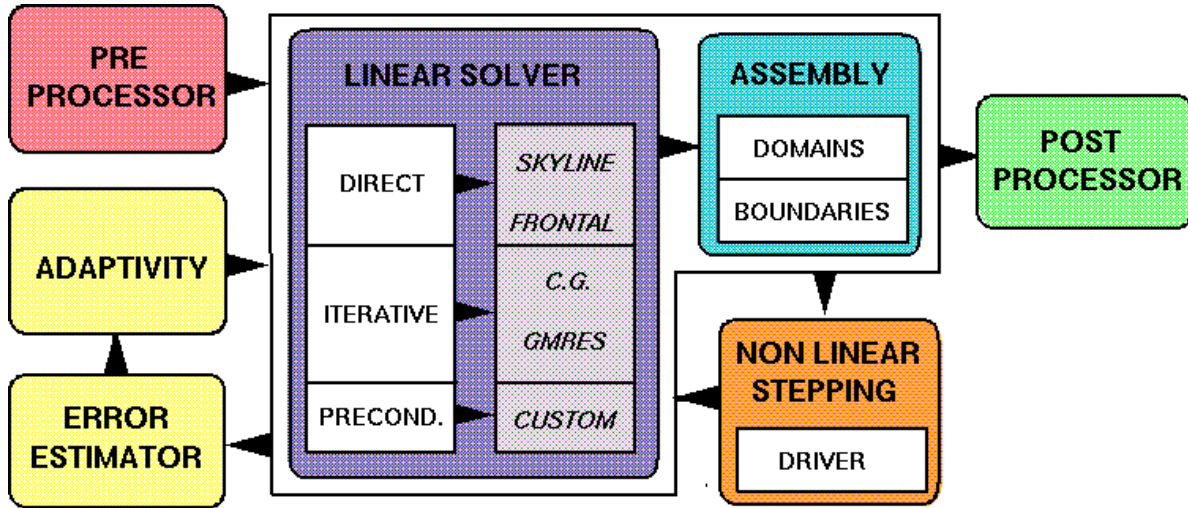


Figure 2: Adaptive solution process with ProPHLEX

error satisfies the specified tolerance, the run is complete. If not, the local, element-wise distribution of the error is processed to determine: the group of elements with the highest relative errors, the critical directions for refinement or enrichment, and whether to refine or enrich a particular element. Once these decisions have been made the mesh adaptor is accessed to modify the computational mesh and database in preparation for the next adaptive solution pass.

The remainder of this section presents specific details regarding various mathematical and algorithmic aspects of the ProPHLEX adaptive process. We begin by reviewing the classes of adaptivity h -, p - and hp - which form an integral part of the ProPHLEX core technology, and outline a few of the complexities related to each of these classes. This is followed by a brief overview of the residual error estimation formulation which is used to make decisions about the quality of the numerical solution and basically drive the dynamic adaptive process.

Before proceeding, we will note that for clarity many of the concepts presented on adaptivity are in a two-dimensional context for quadrilateral elements. In most cases, three-dimensional extensions are outlined and these extensions are the basis of the ProPHLEX library.

Adaptivity

The technology forming the basis of the adaptive package in ProPHLEX has extended the concepts employed with classical adaptive methods [5, 6, 2, 13, 14, 16] by introducing anisotropic hp adaptation. For three-dimensional problems, anisotropic adaptation (whether

$$\begin{aligned}
& \int_{\Omega} \left[a_{11} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + a_{12} \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} + a_{13} \frac{\partial u}{\partial z} \frac{\partial v}{\partial x} + a_{21} \frac{\partial u}{\partial x} \frac{\partial v}{\partial y} + a_{22} \frac{\partial u}{\partial y} \frac{\partial v}{\partial y} + a_{23} \frac{\partial u}{\partial z} \frac{\partial v}{\partial y} + \right. \\
& \left. a_{31} \frac{\partial u}{\partial x} \frac{\partial v}{\partial z} + a_{32} \frac{\partial u}{\partial y} \frac{\partial v}{\partial z} + a_{33} \frac{\partial u}{\partial z} \frac{\partial v}{\partial z} + b_1 \frac{\partial u}{\partial x} v + b_2 \frac{\partial u}{\partial y} v + b_3 \frac{\partial u}{\partial z} v + cuv \right] \partial\Omega \\
& + \frac{1}{\epsilon} \int_{\Gamma_u} uv ds + \int_{\Gamma_g} \frac{1}{b} uv ds \\
& = \int_{\Omega} fv \partial\Omega + \frac{1}{\epsilon} \int_{\Gamma_u} \hat{u} v ds + \int_{\Gamma_g} \frac{g}{b} v ds + \int_{\Gamma_t} tv ds
\end{aligned} \tag{2.3}$$

Note, the implementation of this weak form in ProPHLEX simply requires the identification and encoding the a_{ij} coefficients for the volume integrals and the boundary integral coefficients. There is no coding of the actual integrals, i.e., setting up the shape functions, integration rules and weights, calculating jacobians, etc.

3 Adaptivity and Error Estimation

There are a number of classes of adaptive finite element methods currently in use. In general, they can be roughly classified as follows:

- r -methods, in which nodal locations are modified without changing their total number or the mesh topology,
- h -refinement, based on a local change of the mesh density without changing the order of interpolation. This can be accomplished by remeshing or local subdivision of selected elements,
- p -enrichment, in which the order of interpolation is increased globally or locally to improve the accuracy, and
- hp -methods, in which both the mesh density and the order of interpolation are adapted to minimize the error.

ProPHLEX represents a general approach to the development and implementation of an hp -version of the finite element method. ProPHLEX presents, in particular, adaptive hp methods in which a strategy is developed for choosing local mesh sizes and polynomial degrees so as to reduce local (element-wise) errors in appropriate norms below some preset tolerance. In this way, an optimal mesh is attained in the sense that an attempt is made to produce an hp distribution with the least number of degrees of freedom required to deliver a solution with a given accuracy.

The adaptive methodology in ProPHLEX is basically a three step process (see Figure. 2). First, an estimate of the error is computed using the residual error estimator. If the global

or combinations of the two on the remaining portion:

$$u + b(x, y, z) \frac{\partial u}{\partial n} = g(x, y, z) \quad \text{on } \Gamma_g .$$

The sum of the pieces of the boundary equals the total boundary:

$$\overline{\Gamma_u \cup \Gamma_t \cup \Gamma_g} = \partial\Omega$$

where $\partial\Omega$ is the set of points on the boundary of the domain Ω . Note, If the solution is vector-valued, there must be an equation for each component of the solution similar to those for the scalar case.

Weak Formulation

A variational formulation of Eq. 2.1 is obtained in the standard fashion. Multiply the differential equation by a test function, $v = v(x, y, z)$ (note if u is a vector, v is also a vector with the same number of components), integrate over the domain Ω , and use the Green Gauss divergence theorem to integrate by parts. The resulting weak form is:

$$\begin{aligned} & \int_{\Omega} \left[a_{11} \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + a_{12} \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} + a_{13} \frac{\partial u}{\partial z} \frac{\partial v}{\partial x} + a_{21} \frac{\partial u}{\partial x} \frac{\partial v}{\partial y} + a_{22} \frac{\partial u}{\partial y} \frac{\partial v}{\partial y} + a_{23} \frac{\partial u}{\partial z} \frac{\partial v}{\partial y} \right. \\ & \quad \left. + a_{31} \frac{\partial u}{\partial x} \frac{\partial v}{\partial z} + a_{32} \frac{\partial u}{\partial y} \frac{\partial v}{\partial z} + a_{33} \frac{\partial u}{\partial z} \frac{\partial v}{\partial z} + b_1 \frac{\partial u}{\partial x} v + b_2 \frac{\partial u}{\partial y} v + b_3 \frac{\partial u}{\partial z} v + cuv \right] \partial\Omega \\ & = \int_{\Omega} f v \partial\Omega + \int_{\partial\Omega} \left(\left[a_{11} \frac{\partial u}{\partial x} + a_{12} \frac{\partial u}{\partial y} + a_{13} \frac{\partial u}{\partial z} \right] n_x + \left[a_{21} \frac{\partial u}{\partial x} + a_{22} \frac{\partial u}{\partial y} + a_{23} \frac{\partial u}{\partial z} \right] n_y \right. \\ & \quad \left. \left[a_{31} \frac{\partial u}{\partial x} + a_{32} \frac{\partial u}{\partial y} + a_{33} \frac{\partial u}{\partial z} \right] n_z \right) v ds \end{aligned} \tag{2.2}$$

Introducing the boundary conditions into Eqn 2.2, using a standard penalty approach for the Dirichlet conditions, gives the final variational form which can be directly implemented within the problem formulation portion of the code:

- The equation is in three spatial dimensions, x , y , and z . These three independent variables represent coordinates in a common Cartesian system.
- The dependent variable, u , is the solution to the equation. The variable, u , may be scalar or vector with several components. These components can represent the spatial components of an attribute of the problem being solved, for example: the x -, y -, and z -components of a displacement field for an elasticity problem. However, the components of u do not have to represent components of some sensible vector. For a compressible fluids problem, for example, the solution vector has five components: the density, the x -, y -, and z -components of momentum, and the energy. Taken together, the x -, y -, and z -components of momentum represent the momentum vector, but any other combinations of components of u have no physical meaning. Notice that if u is a vector, then Eq. (2.1) represents a system of equations (an equation for each component of u). Each equation may be significantly different from the others, especially if the solution components differ in physical interpretation (as in the fluids problem).
- The coefficients (a_{11} , a_{12} , a_{13} , a_{21} , a_{22} , a_{23} , a_{31} , a_{32} , a_{33} , b_1 , b_2 , b_3 , c and f) in Eq. (2.1) may be constants, some may be zeros, some may depend explicitly on time, or may even be functions of x, y, z or the solution, u . If u is a vector, then f is also a vector (possibly of functions) with the same number of components. Additionally, the other coefficients are then matrices, each entry of which may be a different constant or function.

For transient problems, a temporal term(s) may be included with each component(s) of the PDE.

Boundary Conditions

To simplify the discussion of boundary conditions somewhat, let us assume that Eq. 2.1 is elliptic. This implies that boundary conditions on each component of the solution must be specified at every point on the domain boundary. The actual form of the boundary conditions, however, may be quite general and each equation may contain virtually any combination of the solution components and their first partial derivatives. (Also note that for transient problems, the boundary conditions may be a function of time).

For a scalar problem, for example, the value of the solution along one portion of the boundary could be specified:

$$u = \hat{u}(x, y, z) \text{ on } \Gamma_u ,$$

the partial derivative of u with respect to the direction normal to the boundary along another portion:

$$\frac{\partial u}{\partial n} = t(x, y, z) \quad \text{on } \Gamma_t ,$$

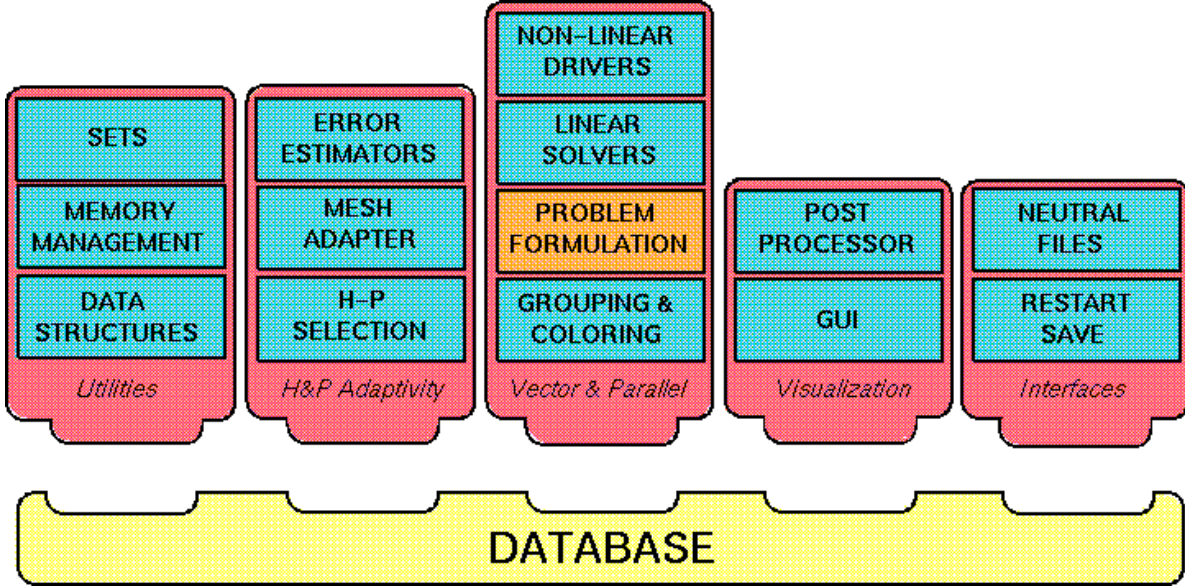


Figure 1: Modular structure of the ProPHLEX library

2 The ProPHLEX Approach to Modeling PDEs

The Differential Equation

The ProPHLEX library is designed to model systems of second order partial differential equations (PDEs) defined on a closed domain. The general form of the differential equation(s) representing the problem must look like:

$$\begin{aligned}
 & -\frac{\partial}{\partial x} \left(a_{11} \frac{\partial u}{\partial x} + a_{12} \frac{\partial u}{\partial y} + a_{13} \frac{\partial u}{\partial z} \right) - \frac{\partial}{\partial y} \left(a_{21} \frac{\partial u}{\partial x} + a_{22} \frac{\partial u}{\partial y} + a_{23} \frac{\partial u}{\partial z} \right) \\
 & - \frac{\partial}{\partial z} \left(a_{31} \frac{\partial u}{\partial x} + a_{32} \frac{\partial u}{\partial y} + a_{33} \frac{\partial u}{\partial z} \right) + b_1 \frac{\partial u}{\partial x} + b_2 \frac{\partial u}{\partial y} + b_3 \frac{\partial u}{\partial z} + cu = f \quad (2.1)
 \end{aligned}$$

where various coefficients in the expansion may be equal to zero. The region in space where a solution is being sought, the domain Ω , is a three-dimensional volume with a boundary denoted as $\partial\Omega$. Specifically, Ω may be envisioned as a set of points inside the domain and $\partial\Omega$ as the set of points on its boundary. The boundary, $\partial\Omega$, may be composed of several different portions on which different types of boundary conditions apply. In subsequent discussions we will refer to a portion of $\partial\Omega$ by the letter Γ , possibly with a representative subscript.

Before introducing the boundary conditions, a few important observations are in order:

[10, 17], and by many others. Importantly, a-posteriori error estimators are often used not only to estimate the error in the numerical solution, but also to drive an automated mesh refinement process with the objective of reducing the error. Extensive reviews of adaptive finite element methods have been compiled by Oden and Demkowicz [15, 16], Noor and Babuška [12] and, in the more general context of automation in computational mechanics, by Tworzydło and Oden [21].

In this paper we present an overview of *ProPHLEX*, a finite element based environment designed to serve as a unique tool for building customized software for modeling complex physical systems, (see figure 1). The physics independent finite element kernel in ProPHLEX is based on a highly advanced formulation of the *hp*-adaptive finite element method. The advantage of this approach, when combined with state-of-the-art residual error estimation to drive the adaptation process, is that, while conventional FEM's can provide only algebraic rates of convergence, an adaptive *hp*-method can result in exponential rates of convergence [19]. In practice, this translates into near-optimum meshes and high-accuracy results for a broad class of problems. As a result of the residual error estimation process, the user also obtains a mathematically correct measure of the overall quality of the solution, thus improving reliability of the FEM analysis.

The ProPHLEX development environment includes an application independent library and a user customizable application template (indicated by the *Problem Formulation* box in figure 1). The finite element template encompasses problem classes which may be linear or nonlinear, steady or unsteady, single component or vector valued, and tightly coupled or weakly coupled. In general, ProPHLEX is applicable to any class of physical phenomena which may be described by a system of linear or nonlinear second order partial differential equations and boundary conditions. User applications are generated through modest customization/editing of a small group of template FORTRAN and C modules which describe the system of equations and boundary conditions. In addition to this basic customization, options are also available for modifying the GUI, post processing, and error estimation modules. After customization, the template is linked with the ProPHLEX library to form the application. Using this application any number of simulations may be performed by supplying a computational mesh, a consistent set of boundary conditions and appropriate material data.

The remainder of this paper presents a general overview of a number of the key components of the ProPHLEX kernel library and the underlying dynamic data base. In particular, details are provided regarding; the general system of PDE's modeled, theoretical background on the residual error error estimation formulation, a review of *hp*-adaptive finite element methods, and a brief overview of the graphics and postprocessing. This is followed by a few selected results, from a solid mechanics application, which are presented to illustrate the practical potential of *hp*-adaptive methods. The paper concludes with a synopsis of the current and future directions of development for the ProPHLEX *hp*-adaptive finite element technology.

ProPHLEX – An *hp*-Adaptive Finite Element Kernel For Solving Coupled Systems of Partial Differential Equations in Computational Mechanics

T.J. Liszka, W.W. Tworzydło, J.M. Bass,
S.K. Sharma, T.A. Westermann, and B.B. Yavari

Computational Mechanics Company, Inc.
7701 N. Lamar, Suite 200
Austin, TX 78752.
<http://www.comco.com>

Abstract

This paper presents an overview of the basic design and architecture of the ProPHLEX *hp*-adaptive finite element kernel. ProPHLEX was designed to be a commercial, robust implementation of *hp*-adaptivity driven by residual error estimation with the primary goal of being physics independent and computationally efficient on an wide array of computer hardware platforms. ProPHLEX can solve virtually any class of engineering problems which may mathematically formulated as a system of linear or nonlinear second order partial differential equations and associated boundary conditions. It has been applied to compressible and incompressible fluid dynamics, linear and non-linear solid mechanics, heat flow problems, as well as semiconductor device simulation. Examples of ProPHLEX customization for linear and nonlinear solid mechanics are presented.

1 Introduction

The Finite Element Method (FEM) has been widely used in the analysis of fluid, electromagnetic, acoustic, and mechanical systems for about three decades now. The field of computational mechanics has come to rely heavily on this technique, and gradually the FEM is becoming the most popular analysis procedure within various fields of design. In the practical application of FEMs, two basic questions are often asked: *What is the accuracy of the numerical solution?* and *What is an optimal mesh for a required level of accuracy?* For a broad class of problems, the answers to these questions are provided by *a-posteriori* error estimates and adaptive computational methods.

The construction of *a-posteriori* error estimates for finite element approximations of boundary value problems has been popularized by Babuška and Rheinboldt and their colleagues [3, 4, 5, 11], Bank [7] and Bank and Weiser [8], Oden, Demkowicz and colleagues